Kathmandu University
Department of Computer Science and Engineering
Dhulikhel, Kavre

Lab Work
"Lab-6"

[Code No: COMP-342]

Submitted by:

Bibhushan Saakha [41]

Submitted to:

Mr Dhiraj Shrestha
Department of Computer Science and Engineering

Submission Date:
2024-06-07

Questions:

1. Implement the following 3D transformations using the 3D shapes provided by Opengl:
    - Translation
    - Rotation
    - Shearing
    - Scaling

2. Implement the Perspective Projection

Code:

```python
import glfw
from OpenGL.GL import *
from OpenGL.GLU import *
import numpy as np

win_width, win_height = 800, 800

def translate(points, tx, ty, tz):
    translation_matrix = np.array([[1, 0, 0, tx],
                                   [0, 1, 0, ty],
                                   [0, 0, 1, tz],
                                   [0, 0, 0, 1]])
    return apply_transformation(points, translation_matrix)

def rotate(points, angle, axis):
    rad = np.radians(angle)
    cos_a = np.cos(rad)
    sin_a = np.sin(rad)
    if axis == 'x':
        rotation_matrix = np.array([[1, 0, 0, 0],
                                    [0, cos_a, -sin_a, 0],
                                    [0, sin_a, cos_a, 0],
                                    [0, 0, 0, 1]])
    elif axis == 'y':
        rotation_matrix = np.array([[cos_a, 0, sin_a, 0],
                                    [0, 1, 0, 0],
                                    [-sin_a, 0, cos_a, 0],
                                    [0, 0, 0, 1]])
    elif axis == 'z':
        rotation_matrix = np.array([[cos_a, -sin_a, 0, 0],
                                    [sin_a, cos_a, 0, 0],
                                    [0, 0, 1, 0],
                                    [0, 0, 0, 1]])
    return apply_transformation(points, rotation_matrix)

def scale(points, sx, sy, sz):
    scaling_matrix = np.array([[sx, 0, 0, 0],
                               [0, sy, 0, 0],
                               [0, 0, sz, 0],
                               [0, 0, 0, 1]])
    return apply_transformation(points, scaling_matrix)

def shear(points, shx, shy, shz):
    shearing_matrix = np.array([[1, shx, shz, 0],
                                [shy, 1, shz, 0],
                                [shx, shy, 1, 0],
                                [0, 0, 0, 1]])
    return apply_transformation(points, shearing_matrix)

```

```python
def apply_transformation(points, matrix):
    transformed_points = []
    for x, y, z in points:
        vec = np.array([x, y, z, 1])
        result = matrix @ vec
        transformed_points.append((result[0], result[1], result[2]))
    return transformed_points

def draw_cube(vertices):
    edges = [
        (0, 1), (1, 2), (2, 3), (3, 0),
        (4, 5), (5, 6), (6, 7), (7, 4),
        (0, 4), (1, 5), (2, 6), (3, 7)
    ]

    glBegin(GL_LINES)
    for edge in edges:
        for vertex in edge:
            glVertex3fv(vertices[vertex])
    glEnd()

def draw_grid():
    glColor4f(0.25, 0.25, 0.25, 0.25)
    glLineWidth(1)
    glBegin(GL_LINES)

    grid_range = np.arange(-10, 11, 1)

    for i in grid_range:
        glVertex3f(i, -10, 0)
        glVertex3f(i, 10, 0)
        glVertex3f(-10, i, 0)
        glVertex3f(10, i, 0)
        glVertex3f(0, i, -10)
        glVertex3f(0, i, 10)
        glVertex3f(i, 0, -10)
        glVertex3f(i, 0, 10)
        glVertex3f(-10, 0, i)
        glVertex3f(10, 0, i)
    glEnd()

def main():
    if not glfw.init():
        return

    window = glfw.create_window(win_width, win_height, "3D Transformations", None, None)
    if not window:
        glfw.terminate()
        return

```

```python
glfw.make_context_current(window)

    glViewport(0, 0, win_width, win_height)
    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    gluPerspective(45, win_width / win_height, 0.1, 50.0)

    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()
    gluLookAt(0, 0, 20, 0, 0, 0, 0, 1, 0)

    cube_vertices = [
        (-1, -1, -1), (1, -1, -1), (1, 1, -1), (-1, 1, -1),
        (-0.5, -0.5, 0.5), (0.5, -0.5, 0.5), (0.5, 0.5, 0.5), (-0.5, 0.5, 0.5)
    ]

    while not glfw.window_should_close(window):
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)

        glPushMatrix()

        draw_grid()

        # Draw original cube
        glColor3f(1, 1, 1)  # White color

        draw_cube(rotate(rotate(cube_vertices,-30,'z'),30,'x'))


        glColor3f(1, 0, 0)  # Red color

        first_vertices = scale(rotate(translate(cube_vertices,-2,-2,0),-30,'y'),2,2,2)
        draw_cube(first_vertices)

        transformed_vertices = scale(rotate(translate(cube_vertices, 2, 2, 0), 30, 'y'),2,2,2)

        glColor3f(0, 0, 1)  # Blue color
        draw_cube(transformed_vertices)

        glColor3f(0.5,0.5,0.5) # Grey color
        draw_cube(translate(shear(cube_vertices,0,0,1),-3,3,0))


        glPopMatrix()

        glfw.swap_buffers(window)
        glfw.poll_events()

    glfw.terminate()

if __name__ == "__main__":
    main()
```

This Python program uses GLFW and OpenGL to perform 3D transformations, including translation, rotation, scaling, and shearing, on a cube. It defines functions for each transformation, applying them to a set of vertices representing the cube. Additionally, the program sets up a perspective projection and renders both the original and transformed states of the cube within a rendering loop. A grid is also visualized with low transparency and thin lines to provide spatial reference. This setup demonstrates the effective application of 3D transformations and projection in a real-time graphics environment.

Outputs: